

Data Acquisition, Processing and Mining for AI

Week 07 — Clustering

Silviu Maniu

Features for mining

acquire → understand → clean → mine

- Cleaning choices (what we drop, flag, or impute) **change the result** of mining
- First mining step: **clustering**, finding groups in the data without labels
- Before clustering: **turning the raw data into numbers**

A title ending in (*) marks **optional** material: context or an extension, dropped first if we run short.

Featurization

- Every algorithm this week and next takes a table of $n \times p$ numbers: n rows and p numeric columns
- First question: what does one row represent? A trip, a repository, a film, a taxi zone. This is a choice, and it decides what the clusters represent
- Raw project data is often text and categories: titles, languages, tags
- **Featurization**: building that numeric table from the raw columns. Which columns we include and how we encode them is a modelling choice

One-hot and counts

language	stars	open_issues		stars	issues	C++	Python	Rust
C++	31 000	220	→	31 000	220	1	0	0
Rust	36 000	380		36 000	380	0	0	1
Python	15 000	45		15 000	45	0	1	0

```
pd.get_dummies(repos, columns=["language"], dtype=int)
```

- **One-hot encoding:** one 0/1 column per category. Numeric columns (counts, amounts) are kept as they are
- **Each (distinct) category becomes a column:** 500 languages means 500 columns, nearly all zero. Group rare categories into **other** before encoding
- **Never encode categories as 1, 2, 3.** That tells the distance formula that Python (2) is between C++ (1) and Rust (3)

Text to numbers

title	fast	in	json	library	parser	rust
"fast json parser"	1	0	1	0	1	0
"json parser in rust"	0	1	1	0	1	1
"fast rust parser"	1	0	0	0	1	1
"a json library"	0	0	1	1	0	0

- **Bag of words**: one column per distinct word, storing how often the word appears in each text. **Word order is lost**
- json appears in 3 of 4 titles, so it **is not discriminative**. **TF-IDF** gives lower weight to common words: term frequency $\times$ $\log(\text{documents} / \text{documents containing the word})$
- Here $\text{idf}(\text{json}) = \ln(4/3) = 0.29$ while $\text{idf}(\text{rust}) = \ln(4/2) = 0.69 \rightarrow$ **rarer words get higher weight**

```
TfidfVectorizer().fit_transform(titles)    # reference (scikit-learn)
```

- Take two repos that differ by 10 000 stars and 3 open issues:

$$d = \sqrt{10\,000^2 + 3^2} = 10\,000.0004$$

- The stars make up **99.99 %** of the squared distance. **The issue count has almost no effect**
- **Standardising**: for each column, subtract its mean and divide by its standard deviation, so **all columns are on the same scale**
- **Counts** with long tails (stars, views, citations): take the **log first**, then standardise
- **Measurements** (miles, minutes, dollars): standardise **as they are**

```
Xt = (feat - feat.mean()) / feat.std()
```

Clustering as EDA

Clustering

- **Clustering**: split the rows into groups so that rows in the same group are **close to each other** and far from the other groups.
- **No labels!**= no right answer is given
- Clustering is used for **exploration**: finding structure in data before we know what groups exist
- Each cluster is **a candidate group**. To see what it means, **look at its summary statistics and its rows**

Distances

- Clustering only sees **the distances between rows**. The choice of distance decides which rows count as close
- **Euclidean**: straight-line distance, $\|x - y\|$. Depends on the **size of the values**
- **Cosine**: the angle between two rows, ignoring their total length, $\cos(x, y) = x \cdot y / (\|x\| \|y\|)$.
- Use **cosine for text**, and **Euclidean for scaled measurements**

text	json	rust	Euclidean to A	cosine with A
A	2	1		
B (longer)	20	10	20.1	1.00
C diff. mix	1	2	1.4	0.80

Two texts (same topic, 3 words and 30 words) are **far apart in Euclidean distance but close in cosine**

k-means

The objective

- Fix the number of clusters k . Pick k cluster centres (**centroids**) $\mu_1 \dots \mu_k$ and assign each point to one of them. Minimise

$$J = \sum_{i=1}^n \|x_i - \mu_{c(i)}\|^2$$

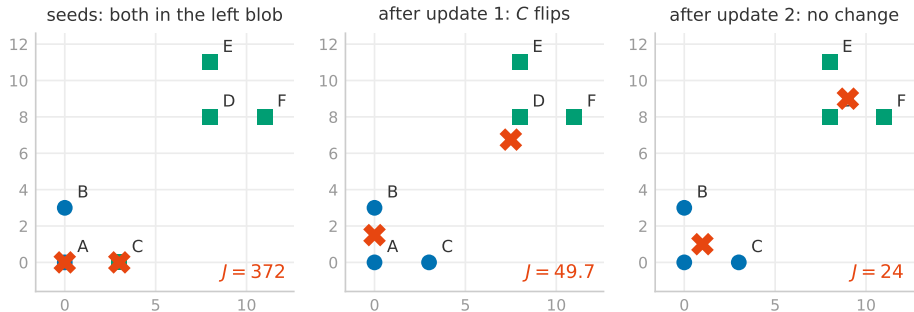
the **inertia**: the sum of squared distances from each point to **its assigned** centroid

- Finding the partition that minimises J over **all** possible partitions is **NP-hard**
- In practice we use a **heuristic**: it finds **a** good partition, not necessarily the best one
- k is a parameter chosen in advance.** Minimising J cannot choose k , because adding more clusters always lowers J

Lloyd's algorithm

- Start with k initial centroids, then repeat two steps:
 1. **Assign**: put each point in the cluster of its **nearest** centroid
 2. **Update**: move each centroid to the **mean** of the points in its cluster
- **Stop when no point changes cluster**
- That is the entire k -means algorithm

By hand: iteration 1



- Six points, $k = 2$, starting centroids $\mu_1 = (0, 0)$ and $\mu_2 = (3, 0)$ (both placed in the left group on purpose). We use squared distances throughout
- **Assign** (squared distance to μ_1 / to μ_2): A 0/9, B 9/18, C 9/0, D 128/89, E 185/146, F 185/128 $\rightarrow$ clusters $\{A, B\}$ and $\{C, D, E, F\}$, with inertia $J = 372$
- **Update** (mean of each cluster): $\mu_1 = (0, 1.5)$, $\mu_2 = (7.5, 6.75)$

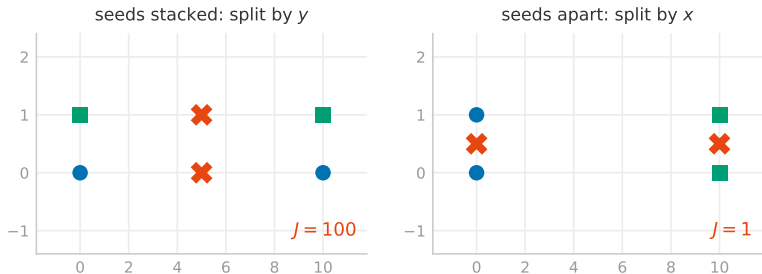
By hand: iteration 2, and the stop

- **Assign** using the new centroids: C has squared distance 11.25 to μ_1 and 65.81 to $\mu_2 \rightarrow C$ moves to the left cluster. All other points stay. $J = 49.69$
- **Update**: $\mu_1 = (1, 1)$ (mean of A, B, C), $\mu_2 = (9, 9)$ (mean of D, E, F)
- **Third pass**: assign again; every point stays in the same cluster $\rightarrow$ **stop**, with final inertia $J = 24$
- Inertia drops at each step: $372 \rightarrow 49.69 \rightarrow 24$. The third pass moves nothing, which is how we know the algorithm has converged

Convergence

- Neither step can increase J : assigning a point to its closest centroid can only reduce its squared distance, and the cluster mean is the exact point that minimises the sum of squared distances within a cluster
- Because J never increases and there are finitely many ways to partition n points, the algorithm must stop (usually in a few passes)
- However, it stops at a local minimum: a grouping that no single step can improve, which may not be the best overall
- Lloyd proposed it in 1957 at Bell Labs for signal quantisation (published in 1982); MacQueen coined the name “ k -means” in 1967

A local optimum (*)



- Four points at (0, 0), (0, 1), (10, 0), (10, 1) in two clear groups
- Starting centroids at (0, 0) and (0, 1) (**stacked** on the left) split the points **by y**: each centroid takes one point from each side. Stops immediately with $J = 100$
- Starting centroids at (0, 0) and (10, 0) split **by x** (left vs. right) and stop with $J = 1$
- **Same data and same algorithm, 100× higher inertia** = the result depends on the starting centroids

Initialization: *k*-means++

- Random starting centroids are often bad → spread them out
- *k*-means++: pick the first centroid uniformly at random; pick each next one with probability proportional to $D(x)^2$, the squared distance to the nearest centroid already chosen
- On the six points, after starting at A, the squared distances are [0, 9, 9, 128, 185, 185] → the right group has $498/516 = 96.5\%$ of the probability.
- Guarantees an expected J within $\mathcal{O}(\log k)$ of the optimum (the only theoretical guarantee of *k*-means...)
- Even with *k*-means++, run the algorithm several times from different seeds and keep the run with the lowest inertia

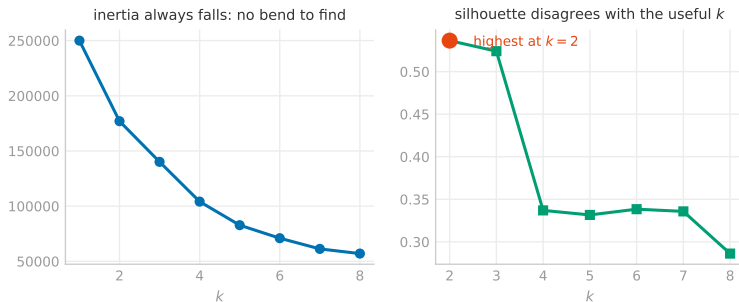
```
km_pp = KMeans(n_clusters=3, init="k-means++", n_init=10).fit(X)
```

```
# d2[i, j]: squared distance from point i to centroid j (n x k)
d2 = ((X[:, None, :] - centroids[None, :, :]) ** 2).sum(-1)
labels = d2.argmin(1)
inertia = float(d2.min(1).sum())
```

- Full assignment step, **no loop over rows**: broadcasting computes the $n \times k$ table of squared distances in one expression
- **Vectorisation**: NumPy runs the loop in C over contiguous arrays
- Each pass costs $\mathcal{O}(nkp)$ time: linear in the number of rows, clusters, and columns $\rightarrow k$ -means scales to large tables
- **Computing inertia**: inside the loop, centroids move after each assignment. When the loop stops, **assign one last time** so the inertia matches the returned centroids

Choosing k

The elbow



- A possible way: inertia against k and look for a **bend** (an elbow), the point where adding more clusters gives only a small drop
- On the taxi features ($k = 1 \dots 8$), inertia drops smoothly from 250 000 to 57 000. **There is no clear bend**, which is typical on real data
- Schubert (2023) argues the elbow rule should **not be used**. We often see a bend in a smooth curve because we expect one

Silhouette

- For each point, compare a (mean distance to **its own** cluster) and b (mean distance to the **nearest other** cluster):

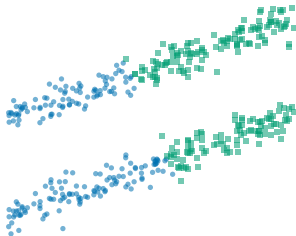
$$s = \frac{b - a}{\max(a, b)} \in [-1, 1]$$

Near 1: well inside its cluster; **near 0**: on the boundary between two clusters; **negative**: closer to another cluster than to its own

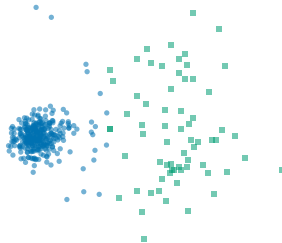
- On the six points, for $B = (0, 3)$: $a = 3.62$, $b = 10.94$, $s = 0.67$; the mean over all six points is **0.70**
- On the taxi features, the mean silhouette is **highest at $k = 2$** (0.54), which only separates airport trips from the rest. **A high silhouette means compact groups, not useful ones**
- Rousseeuw (1987) proposed the silhouette as **a diagnostic plot**, not a maximisation criterion for k

Limits of k -means

two elongated clusters → cut across



one dense, one sparse → cut across



- Assigning each point to the nearest centroid cuts space with **straight boundaries**, so k -means expects **round clusters of similar size and density**. Real groups often have other shapes
- **Outliers pull centroids towards them**: a single extreme trip (**222 167 miles**) can take an entire cluster for itself
- We must **fix k in advance** + and every point **is forced into a cluster**

Other shapes

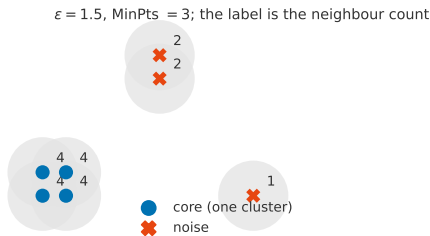
DBSCAN: clusters as dense regions

- Takes two parameters: a radius ε and a minimum neighbour count **MinPts**. Every point falls into one of three types:
 - **core**: has at least MinPts points within distance ε (including itself)
 - **border**: fewer than MinPts within ε , but lies within ε of a core point
 - **noise**: neither core nor border; **belongs to no cluster**
- Clusters **grow by connecting nearby core points**, so they can follow **any shape**
- Advantages over k -means: **no need to choose k** , and **isolated points are labelled as noise**
- Ester, Kriegel, Sander & Xu (1996). Main difficulty: **choosing ε is hard**, especially when density varies or in many dimensions

```
lab = DBSCAN(eps=eps, min_samples=5).fit_predict(noisy)
```

Reference in scikit-learn: `min_samples` is MinPts (point itself included); **noise gets label -1** .

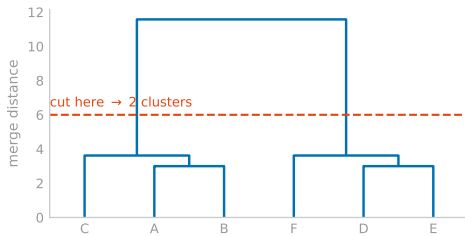
DBSCAN by hand (*)



- Seven points with $\epsilon = 1.5$ and $\text{MinPts} = 3$. Each point is labelled with **how many points lie within distance ϵ** (including itself)
- The four points in the unit square each have **4** neighbours within $\epsilon \rightarrow$ all four are core points and form one cluster
- (5, 5) and (5, 6) have **2** each (below $\text{MinPts} = 3$), and (9, 0) has only **1** $\rightarrow$ all three are labelled as **noise**
- **If we lower MinPts to 2, the pair (5, 5) and (5, 6) becomes a second cluster**

Hierarchical clustering

- **Agglomerative** (bottom-up): start with each point in its own cluster, merge the **two closest clusters**, and repeat until all points are in one cluster
- **Linkage** defines the distance between two clusters: **single** (closest pair of points), **complete** (furthest pair), or **average** (mean over all pairs)
- The **dendrogram** plots every merge and the distance at which it happened. **Cut the tree at a chosen height** to get clusters, without fixing k in advance
- Needs $\mathcal{O}(n^2)$ **memory** for the pairwise distances: works for a few thousand rows, not for 200 000 trips



Evaluating without labels

Evaluating without labels

- Without ground-truth labels, there is **no accuracy score**. Internal metrics (inertia, silhouette) only measure **the geometry of the clusters**, not whether they make sense
- Both inertia and silhouette reward **compact, well-separated balls** → they naturally favour k -means even when the clusters are not meaningful
- **Stability** is a useful check: **re-run with different random seeds, or on a random subsample**. If the clusters change completely, they are an artefact of the run
- **Inspecting the rows** is always needed to see what each cluster represents

Reading a cluster

trip_distance	duration_min	fare_amount	tip_share	pickup_hour	n	what it is
1.82	11.36	12.79	0.31	14.70	27503	short trips
2.23	13.59	15.42	0.03	13.94	17800	not much tip
13.76	38.62	58.40	0.16	13.91	4683	airport run
2.95	1416.80	17.24	0.12	12.93	14	?

- Table shows the feature means of the $k = 4$ clusters on 50 000 trips (clustered on five standardised features). **Three clusters are easy to name**
- In the airport cluster, **28 %** of trips (1 320 rows) have RatecodeID = 2, and **every one of them has a fare of exactly \$70**
- The fourth cluster has only **14 trips**, with a mean duration of **23.6 hours**. **Trips with broken meters!**
- **A cluster that has no “name” often points to a data-quality issue**

Pitfalls

- **Forgetting to scale features:** the column with the largest numbers dominates all distances
- **Euclidean distance on many one-hot columns:** any two different categories are $\sqrt{2}$ apart
- **Running k -means with a single random seed:** always run multiple restarts from different seeds and check that the groups are stable
- **Picking k to fit a story** – choose your criterion for k **before** looking at the clusters
- **Reporting cluster numbers instead of interpreting them:** **six named and described groups** not “we have 6 clusters”

Lab and next week

Today's lab

- M2 due last night: findings, cleaning notebook, data card
- Mandatory exercises: Lloyd's algorithm (slide *In NumPy*), choosing k , and the real trips, naming clusters
- Solved: synthetic “blobs”, the six example points, check against scikit-learn
- Optional: zones by hourly profile, DBSCAN and noise labels

Before end of course

- **mining step**: decide what a row is, build feature table (X), cluster, and **name every group**
- **M3** (pipeline, report) is due **one week after** week 8's session, **defenses in that session**
- Next lecture: **frequent itemsets**

References

- Lloyd, *Least Squares Quantization in PCM*, IEEE Trans. Inf. Theory 28(2), 1982 (written 1957). <https://doi.org/10.1109/TIT.1982.1056489>
- MacQueen, *Some Methods for Classification and Analysis of Multivariate Observations*, 5th Berkeley Symp., 1967 — the name “ k -means”.
- Arthur & Vassilvitskii, *k -means++: The Advantages of Careful Seeding*, SODA 2007. <https://dl.acm.org/doi/10.5555/1283383.1283494>
- Ester, Kriegel, Sander & Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise (DBSCAN)*, KDD 1996. <https://cdn.aaai.org/KDD/1996/KDD96-037.pdf>
- Rousseeuw, *Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis*, J. Comput. Appl. Math. 20, 1987. [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7)
- Schubert, *Stop Using the Elbow Criterion for k -Means*, SIGKDD Explorations 25(1), 2023. <https://doi.org/10.1145/3606274.3606278>
- scikit-learn, *Clustering*: <https://scikit-learn.org/stable/modules/clustering.html>